

Online Bigtable merge compaction

Claire Mathieu¹, Carl Staelin², Neal E. Young^{*3}, and Arman Yousefi^{*4}

¹CNRS, Paris

`cmathieu@di.ens.fr`

²Google Israel Engineering Center, Haifa

`staelin@google.com`

³University of California, Riverside

`neal.young@ucr.edu`

⁴University of California, Los Angeles

`armany@cs.ucla.edu`

Abstract

NoSQL databases are widely used for massive data storage and real-time web applications. Yet important aspects of these data structures are not well understood. For example, NoSQL databases write most of their data to a collection of files on disk, meanwhile periodically *compacting* subsets of these files. A *compaction policy* must choose which files to compact, and when to compact them, without knowing the future workload. Although these choices can affect computational efficiency by orders of magnitude, existing literature lacks tools for designing and analyzing online compaction policies — policies are now chosen largely by trial and error.

Here we introduce tools for the design and analysis of compaction policies for Google Bigtable, propose new policies, give average-case and worst-case competitive analyses, and present preliminary empirical benchmarks.

^{*}Supported by Google research award *A Study of Online Bigtable-Compaction Algorithms*, and NSF grant 1117954.

Introduction — NoSQL databases and BigTable compaction

NoSQL databases provide distributed, reliable, high-volume, real-time data storage. Companies making heavy use of NoSQL systems include Adobe, Ebay, Facebook, GitHub, Meetup, Netflix, and Twitter. At Google, *BigTable* servers support applications such as Gmail, Maps, Search, Crawl, Google+, Analytics, and Base. Published data (most recently from 2006) show over 24,500 BigTable servers, supporting over 1.2 million requests per second and 16 GB/s of outgoing RPC traffic, and holding over a petabyte of data for Google Crawl and Analytics alone [5, §8].

For a general introduction to NoSQL, see [4, 16, 18]. Roughly, NoSQL databases support reads and writes of key/value pairs. Almost all modern NoSQL systems employ a “Log-Structured-Merge” (LSM) architecture: a cache holds recent writes, which are periodically aggregated and pushed to immutable disk files. This is in contrast to traditional DBMSs, which update data files in place, leading to slower insertions and updates. LSM systems organize their files in *levels* by partitioning time into intervals and storing all writes from a particular interval in one level. The most recent level (ending at the current time) is held in the cache. Each remaining level is held on disk, either in a single file or, by a partition of the key space, in multiple files. Periodically, the cache is dumped to disk, creating a new level. (The cache may be dumped for various reasons, not just when it is full.) The time per read grows with the number of levels — a typical read searches the levels, most recent first, checking one file in each level until the desired key is found. To keep the number of levels bounded, contiguous levels are periodically merged. This merge process is referred to as *compaction*. Compaction and read operations together account for a significant fraction of the computing resources used by the system, and can be the main bottleneck [5, §7].

Here we focus on improving the efficiency of compaction and reads. We focus on Google’s BigTable database, but the proposed principles may also be applied to other LSM storage systems, most immediately to those that, like Bigtable, use just one file per level (e.g. Accumulo [13, 15], AsterixDB [1], HBase [15, 8, 14], Hypertable [14, 11], and Spanner [7]). We develop techniques for the design and analysis of compaction policies, analyze new policies using worst-case and average-case competitive analyses, give absolute estimates of optimal costs, and present preliminary benchmarks.

This is the first formal study of online compaction policies that we know of.¹

Formal definition of Bigtable merge compaction (BMC). Formally, for any non-decreasing *read-cost function* $f : \mathbb{R}_+ \rightarrow \mathbb{R}_+$, define BMC_f as follows. The input is a sequence $\mathcal{I} = \langle (\ell_t, r_t) \rangle_t \in (\mathbb{R}_+ \times \mathbb{R}_+)^n$. The algorithm maintains a stack of lengths, initially empty. At time t , the pair (r_t, ℓ_t) is revealed, where r_t is the *read rate* and ℓ_t is the length at time t (representing the length of the new disk file created from a cache dump). The length ℓ_t is inserted at the top of the stack. The algorithm A then chooses a *compaction*: it selects some contiguous sequence of lengths at the top of the stack, then adds them to get a single new length L_t , which replaces them in the stack. At time t , the *merge cost* is L_t ; the *read cost* is $r_t f(k_t)$, where k_t is the stack size after the compaction at time t . The output, called a *schedule*, is the sequence σ of n compactations. The cost of σ on \mathcal{I} , denoted $\sigma(\mathcal{I})$ or $A(\mathcal{I})$, is $\sum_{t=1}^n L_t + r_t f(k_t)$. Figure 1 shows an example schedule.

Current practice at Google is to constrain the number of levels to a parameter K , otherwise ignoring read costs. We use $\text{BMC}_{\leq K}$ to denote this special case of BMC_f , which is obtained by taking $f(k) = 0$ if $k \leq K$ and $f(k) = \infty$ otherwise. The parameter K is tuned manually on a per-table basis, based on historical workload. This is reliable, but slow, costly, and inflexible. To

¹Ghosh et al. study the related but quite different problem of performing a single offline compaction via a sequence of merges, given a constraint on the number of files that can be merged at once. That problem is NP-hard [9]. As far as we know, NoSQL is not yet studied in the large literature on *external-memory* algorithms [2, 19].

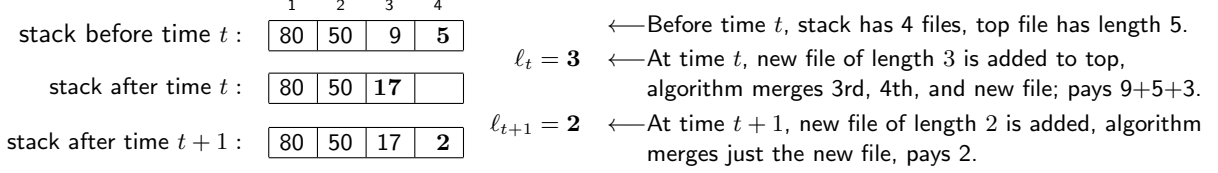


Figure 1: Steps t and $t + 1$ of a BMC_f schedule.

explore compaction policies that instead adjust stack size *automatically*, we also consider LINEAR BMC , which is BMC_f with $f(k) = k$.

For more intuition about the combinatorial structure of BMC_f , note that the restriction of $\text{BMC}_{\leq K}$ to *uniform* instances (those with $(\ell_t, r_t) = (\bar{\ell}, \bar{r})$ for all t) is essentially the *egg-dropping puzzle* with n floors and K eggs [17, Thm. 2] ([3] gives other applications). The restriction of LINEAR BMC to uniform instances is equivalent to *lopsided alphabetic binary coding* [6, 10, 12]. We encourage the reader to try solving a uniform instance of $\text{BMC}_{\leq K}$ with n unit lengths and, say, $K = 1$ and then $K = 2$. Uniform instances are already combinatorially non-trivial; the general cases with non-uniform inputs are significantly more complicated.

Throughout, $X \sim Y$ means $X = (1 \pm o(1))Y$, where $o(1)$ denotes a quantity that tends to zero as $n = |\mathcal{I}|$ tends to infinity. *With high probability* means with probability $1 - o(1)$, and $[i, j]$ denotes $\{i, i+1, \dots, j\}$. $\mathcal{I}[i, j]$ denotes $(\ell_i, r_i), (\ell_{i+1}, r_{i+1}), \dots, (\ell_j, r_j)$. A compaction algorithm A is *online* if its choice at time t depends only on $\mathcal{I}[1, t]$. A is *c-competitive* if $A(\mathcal{I}) \leq c \text{OPT}(\mathcal{I})$ for every instance \mathcal{I} . Given a random instance \mathcal{I} , A is *c-competitive in expectation* if $\mathbb{E}_{\mathcal{I}}[A(\mathcal{I})] \leq c \mathbb{E}_{\mathcal{I}}[\text{OPT}(\mathcal{I})]$, and *asymptotically 1-competitive in expectation* if $\mathbb{E}_{\mathcal{I}}[A(\mathcal{I})] \sim \mathbb{E}_{\mathcal{I}}[\text{OPT}(\mathcal{I})]$.

Summary of main theorems

Theorem 1 (worst-case analysis of $\text{BMC}_{\leq K}$). *There is an online algorithm (called BRB) for $\text{BMC}_{\leq K}$ that is K -competitive. No deterministic online algorithm is less than K -competitive.*

Theorem 2 (bijection with binary search trees). *For any instance \mathcal{I} of BMC_f , the schedules σ for \mathcal{I} are isomorphic to the n -node binary search trees T , under a natural cost function. . .*

Theorem 3 (worst-case analysis of LINEAR BMC). *There is an online algorithm for LINEAR BMC that is $O(1)$ -competitive on “read-heavy” instances \mathcal{I} — those s.t. $\ell_t = O(r_t)$ for all t .*

Theorem 4 (average-case analyses). *$\text{BMC}_{\leq K}$ and LINEAR BMC have online algorithms A and B , respectively, that are asymptotically 1-competitive in expectation on random inputs \mathcal{I} with bounded, i.i.d. requests. On such an \mathcal{I} , letting $(\bar{\ell}, \bar{r}) = (\mathbb{E}_{\mathcal{I}}[\ell_t], \mathbb{E}_{\mathcal{I}}[r_t])$ (for all t), for $\text{BMC}_{\leq K}$,*

$$\mathbb{E}_{\mathcal{I}}[A(\mathcal{I})] \sim \mathbb{E}_{\mathcal{I}}[\text{OPT}(\mathcal{I})] \sim \bar{\ell} K n^{1+1/K} / c_K$$

where $c_K = (K + 1)/(K!)^{1/K}$ (so $c_K \rightarrow e$ for large K). For LINEAR BMC ,

$$\mathbb{E}_{\mathcal{I}}[B(\mathcal{I})] \sim \mathbb{E}_{\mathcal{I}}[\text{OPT}(\mathcal{I})] \sim \beta_{\mathcal{I}} n \log_2 n,$$

for $\beta_{\mathcal{I}} = \beta$ such that $1/2^{\beta/\bar{\ell}} + 1/2^{\beta/\bar{r}} = 1$, so $\beta = \Theta(\bar{\ell} + \bar{r}) / \ln(1 + \max(\bar{\ell}/\bar{r}, \bar{r}/\bar{\ell}))$.

Benchmarks. In many applications at Google, the lengths of inserted files (the ℓ_t ’s) follow log-normal distributions. Section 5 presents empirical benchmarks on such distributions. The algorithm from Theorem 1, BRB — balanced rent-or-buy, performs nearly optimally, better (sometimes substantially) than the current default BigTable compaction algorithm (for $\text{BMC}_{\leq K}$).

Techniques. BRB, our K -competitive algorithm for $\text{BMC}_{\leq K}$, is a recursive rent-or-buy scheme that roughly balances the cost incurred in each of the K stack positions. BRB happens to be asymptotically *optimal* on uniform instances. The proof of K -competitiveness is by induction on K . The proof that no algorithm is better than K -competitive uses a non-trivial recursive generalization of the standard rent-or-buy adversary argument.

Offline BMC_f has straightforward dynamic-programming algorithms — $O(n^4)$ time for BMC_f , $O(Kn^3)$ for $\text{BMC}_{\leq K}$, $O(n^3)$ for LINEAR BMC (Corollary 2). Theorem 2 (the bijection with binary trees) is the critical observation that unlocks LINEAR BMC for further analysis. The theorem yields a tree-based lower bound on OPT (Lemma 3) analogous to entropy-based lower bounds for alphabetic codes [10]. The lower bound in turn is used to give a linear-time 2-approximation algorithm for LINEAR BMC (Corollary 3), and to bound OPT in the proof of Theorem 3.

Theorem 2 is also used in the proof of Theorem 4: firstly, to bound optimal solutions for *uniform* instances $\bar{\mathcal{I}}$ (which correspond exactly to optimal binary search trees and alphabetic codes, whose costs are well understood); secondly, to show that, with high probability, random instances \mathcal{I} and uniform instances have the same asymptotic cost.

Remarks. One aspect of compaction not modeled by BMC_f as defined here is that key/value pairs may leave the database, due to expiration, deletion, or redundancy. When a compaction merges several files into one file F , the length of F may be *less than* the length of the merged files. We note without proof that the K -competitive algorithm BRB for $\text{BMC}_{\leq K}$ (and its proof) extend naturally to show K -competitiveness in this more general setting.

It is natural to extend BMC_f to allow so-called *interior merges*, which merge contiguous levels *within* the stack. OPT never uses interior merges, nor does BRB (which remains optimally K -competitive for $\text{BMC}_{\leq K}$ even if interior merges are allowed). But we conjecture that any $O(1)$ -competitive online algorithm for general LINEAR BMC will require interior merges.

We’re conducting further benchmarks using AsterixDB, after which we’ll benchmark on Google BigTable servers. Many theoretical problems remain open. Is BRB asymptotically 1-competitive in expectation on bounded i.i.d. inputs? Is there an $o(K)$ -competitive *randomized* online algorithm for $\text{BMC}_{\leq K}$? Is there an $O(1)$ -competitive online algorithm for general LINEAR BMC?

1 Worst-case competitive analysis of $\text{BMC}_{\leq K}$

Definition of algorithm BRB_K for $\text{BMC}_{\leq K}$ on input \mathcal{I} . For $K = 1$, there is only one possible schedule: at each time t , all files are merged into one. For $K > 1$, BRB_K partitions the times $[1, n]$ into intervals called *phases*. The first phase $[1, 1]$ starts and ends at time 1. Each subsequent phase $[s, s']$ ends with BRB_K merging all files into one file at time s' . To handle the requests in $[s, s' - 1]$ (before the end of the phase), BRB_K runs BRB_{K-1} recursively, ignoring the single file at the bottom of the stack from the previous phase. The phase is as long as possible, subject to the constraint that the cost that BRB_{K-1} incurs during the phase, $\text{BRB}_{K-1}(\mathcal{I}[s, s'])$, is less than $K - 1$ times the cost of the single merge that BRB_K does to end the phase, $\ell[1, s']$. (See (a) in the proof below.)

Theorem 1 (worst-case analysis for $\text{BMC}_{\leq K}$). (i) BRB_K is K -competitive for $\text{BMC}_{\leq K}$.

(ii) No deterministic online algorithm for $\text{BMC}_{\leq K}$ is less than K -competitive.

The proof consists of the two lemmas below.

Lemma 1.1 (Part (i)). *There exists a K -competitive online algorithm for $\text{BMC}_{\leq K}$.*

Proof. Fix an input \mathcal{I} . Let $\mathcal{I}[i, j]$ denote the subsequence $(\ell_i, r_i), \dots, (\ell_j, r_j)$ of \mathcal{I} . Let $\ell[i, j] = \sum_{h=i}^j \ell_h$. For $K = 1$, all algorithms are the same, hence 1-competitive. To complete the proof, for $K > 1$, we show that, for each phase $[s, s']$, during the phase, the cost incurred by BRB_K is at most K times the cost incurred by OPT . First consider any phase that ends with BRB_K merging all files into one (as happens in every phase except maybe the last). During the phase:

- (a) BRB_K chooses s' so $\text{BRB}_{K-1}(\mathcal{I}[s, s' - 1]) < (K - 1) \ell[1, s'] \leq \text{BRB}_{K-1}(\mathcal{I}[s, s'])$.
- (b) BRB_K incurs cost $\text{BRB}_{K-1}(\mathcal{I}[s, s' - 1]) + \ell[1, s']$.
- (c) OPT incurs cost at least $\min \left\{ \frac{1}{K-1} \text{BRB}_{K-1}(\mathcal{I}[s, s']), \ell[1, s'] \right\}$. (This is proven below.)

Bounds (a-c) above imply, by algebra, that BRB_K 's cost during the phase is at most K times OPT 's cost during the phase. The proof of (c) has two cases:

OPT merges all files into one at some time $t \in [s, s']$. For that merge OPT pays $\ell[1, t]$. At each time $t' \in [t + 1, s']$ OPT pays at least $\ell_{t'}$. OPT 's total cost during the phase is at least $\ell[1, s']$.

OPT never merges all files into one during $[s, s']$. Whatever file OPT had at the bottom of the stack at time s remains untouched throughout the phase. Hence, OPT handles $\mathcal{I}[s, s']$ using only $K - 1$ stack slots. By induction, BRB_{K-1} is $(K - 1)$ -competitive on $\mathcal{I}[s, s']$, so OPT 's cost to do so is at least $\text{BRB}_{K-1}(\mathcal{I}[s, s']) / (K - 1)$.

Finally, consider any phase that ends without BRB_K merging all files into one (this must be the final phase). Bound (c) above holds by the same argument. BRB_K 's cost in the phase is $\text{BRB}_{K-1}(\mathcal{I}[s, s'])$ which, by definition of BRB_K , since BRB_K doesn't merge, is less than $(K - 1) \ell[1, s']$. This and (c) imply that BRB_K 's cost during the phase is most $K - 1$ times OPT 's cost. \square

Lemma 1.2 (Part (ii)). *No deterministic online algorithm for $\text{BMC}_{\leq K}$ is less than K -competitive.*

Proof. Fix any deterministic online algorithm A . We will define a $\text{BMC}_{\leq K}$ instance \mathcal{I} such that $A(\mathcal{I}) / \text{OPT}(\mathcal{I})$ is at least $(1 + O(K/L_K)) K$ where $L_K \gg K$ is an arbitrarily large integer. This will prove Part (ii).

The lengths in \mathcal{I} will be *well-separated*, enabling us to use a *max-based cost* in the analysis:

Definition 1.1 (well-separated). *A set of lengths is well-separated (w.r.t. L_K) if every two non-zero lengths in the set differ by a factor of at least L_K . Sequence \mathcal{I} is well-separated if its lengths are.*

Definition 1.2 (max-based cost). *Recall that in the definition of $\text{BMC}_{\leq K}$ merging a collection of files generates a file whose length is the sum of the merged lengths. Modify the definition so that, instead, the merged file's length (and the cost of the merge) is the maximum of the merged files's lengths. The max-based cost (of a merge, or of a schedule) is the cost using this modified definition.*

Lemma 1.3. *For any well-separated sequence \mathcal{I} and any schedule σ , the true cost $\sigma(\mathcal{I})$ is at most $1/(1 - 1/L_K) = 1 + O(1/L_K)$ times its max-based cost $\sigma'(\mathcal{I})$.*

Proof. With the original definition, the length of a file in the stack at any time is the sum $\sum_{t=i}^j \ell_t$ of some interval of lengths in the given instance \mathcal{I} . With the modified definition, the length of the file is instead $\max_{t=i}^j \ell_t$, the maximum length in the interval. Since \mathcal{I} is well separated, $\sum_{t=i}^j \ell_t \leq \max_{t=i}^j \ell_t (1 + 1/L_K + 1/L_K^2 + \dots) = \max_{t=i}^j \ell_t / (1 - 1/L_K)$. \square

To prove the theorem, we construct a well-separated \mathcal{I} for which the max-based cost $\text{OPT}'(\mathcal{I})$ is at most $1/K + O(1/L_K)$ times the true cost $A(\mathcal{I})$ of A on \mathcal{I} .

Before we define the lengths to be used in \mathcal{I} , fix K integers $L_1 \gg L_2 \gg \dots \gg L_K \gg K$, by choosing arbitrarily large $L_K \gg K$, then defining each L_h for $h \in [1, K-1]$ from $\{L_{h+1}, \dots, L_K\}$ via

$$L_h = L_{h+1} L_K^{N_h} \quad \text{where} \quad N_h = \prod_{i=h+1}^K L_i. \quad (1)$$

For each $h \in [1, K]$, define the h -lengths: $w_{h1} \ll w_{h2} \ll \dots \ll w_{hN_h}$ by taking $w_{hi} = L_K^i / L_h$.

Lemma 1.4. (i) The set $\{w_{hi}\}_{h,i}$ of lengths defined above is well separated.

(ii) Each h -length w_{hi} is at most 1, but satisfies $L_h w_{hi} \geq L_K$.

Proof. For any $h \in [1, K]$, the h -lengths are well-separated among themselves. The largest h -length is w_{hN_h} , which (by (1) and Def. of w) is at most $1/L_K$ times the smallest $(h+1)$ -length $w_{h+1,1}$. This implies that the h -lengths are well-separated from the $(h+1)$ -lengths, so the complete set is well-separated. It also implies that each length w_{hi} is at most $w_{K1} = 1$. By inspection, $L_h w_{hi} \geq L_K$. \square

Define the request sequence \mathcal{I} inductively via *phases*. A *1-phase* inserts the next unused 1-length, then repeatedly inserts zeros; it stops when the algorithm merges the 1-length with a larger length or the 1-phase has inserted L_1 zeros. For $h \in [1, K-1]$, an h -*phase* inserts the next unused h -length, then repeatedly does $(h-1)$ -phases; it stops when the algorithm merges the h -length with a larger length or the h -phase has done L_h $(h-1)$ -phases. A K -*phase* reveals inserts the K -length $w_{K1} = 1$, then does L_K $(K-1)$ -phases. The sequence \mathcal{I} is just a single K -phase.

Observe that \mathcal{I} uses exactly one K -length, exactly L_K $(K-1)$ -lengths, at most $L_K L_{K-1}$ $(K-2)$ -lengths, and, for $h \in [1, K]$, at most N_h h -lengths (for N_h from (1)).

For $h \in [1, K]$, let n_h ($\leq N_h$) denote the total number of h -phases in \mathcal{I} . (This depends on the algorithm.) For $i \in [1, n_h]$, let n_{hi} denote the number of $(h-1)$ -phases (or number of zeros if $h=1$) within the i th h -phase. Note $n_K = 1$ and $n_{K1} = L_K$.

Lemma 1.5. The max-based cost of OPT on \mathcal{I} is at most $2 + \frac{1}{K} \sum_{h=1}^K \sum_{i=1}^{n_h} w_{hi} n_{hi}$.

Proof. We show that there exists a schedule of at most the desired max-based cost.

Recall that we have $K+1$ types of lengths in \mathcal{I} : zeros, 1-lengths, 2-lengths, \dots , K -lengths (in order of increasing length). Call zeros *0-lengths*.

Consider K different K -slot schedules $\beta(1), \beta(2), \dots, \beta(K)$, where, for each $b \in [1, K]$, schedule $\beta(b)$ chooses slots according to the following rule: *Given an h -length, if $h < b$, then merge it into slot $h+1$, else merge it into slot h .* That is, slot b receives by $(b-1)$ -lengths and b -lengths; every other length type h goes in its own slot: h (if $h < b-1$) or $h+1$ (if $h > b$).

What is the max-cost of $\beta(b)$ on \mathcal{I} ? Consider the h -lengths ℓ_t with $h \neq b-1$. For such a length, $\beta(b)$ merges the length only with previously merged ℓ -lengths where $\ell \leq h$. Because all ℓ -lengths with $\ell < h$ are smaller than all h -lengths, and h -lengths occur in \mathcal{I} in increasing order, these other lengths are smaller than ℓ_t , so the max-based merge cost is ℓ_t . Hence, the total cost of such merges is at most $\sum_t \ell_t = \sum_{h=1}^K \sum_{i=1}^{n_h} w_{hi}$. Further, since the lengths are well separated, this sum is at most $w_{11}/(1 - 1/L_K) \leq 2$.

Next consider the insertion of any $(b-1)$ -length $\ell_t = w_{b-1,j}$. The max-cost of its merge is the most recently revealed b -length, say w_{bi} . So, the b -length from b -phase i contributes its length to the aggregate max-cost once for each $(b-1)$ -phase that occurs in b -phase i .

In sum, the max-cost of $\beta(b)$ is at most $2 + \sum_{i=1}^{n_b} w_{bi} n_{bi}$. Hence, the max-based-costs of the K schedules $\{\beta(b)\}_b$ are, on average, at most the bound claimed in the lemma. \square

Lemma 1.6. *The cost of A on \mathcal{I} is at least $(1 - 1/L_K) \sum_{h=1}^K \sum_{i=1}^{n_h} n_{hi} w_{hi}$.*

Proof. When a merge occurs at time t , the cost $s_{\sigma_t}^{t+1}$ of the merge is the sum of some interval $\mathcal{I}[i, t]$ of lengths in \mathcal{I} ; say each length in this interval *contributes its value to the merge*. The total contributions of all lengths in \mathcal{I} (to all merges) equals the cost of the schedule.

For $i \in [1, n_1]$, the i th 1-phase reveals 1-length w_{1i} , then n_{1i} zeros. Slot 1 is not emptied before the phase ends, so slot 1 contains w_{1i} until the end of the phase, so each of the n_{1i} zeros causes w_{1i} to contribute to one merge, contributing in total at least $n_{1i} w_{1i}$. For $h > 1$, for $i \in [1, n_h]$, the i th h -phase reveals h -length w_{hi} , then does n_{hi} $(h-1)$ -phases. Slot h is not emptied before the h -phase ends, so w_{hi} is contained in a slot in $[1, h]$ until the end of the h -phase. Each $(h-1)$ -phase j in the i th h -phase either (a) ends with a merge that empties slot $h-1$, which must cause w_{hi} to contribute to that merge, or (b) *times out* — that is, $(h-1)$ -phase j does $n_{h-1,j} = L_{h-1}$ iterations. Let τ_{hi} be the number of $(h-1)$ -phases in the h -phase that time out, so that length w_{hi} 's contributions total at least $(n_{hi} - \tau_{hi})w_{hi}$. Summing over the lengths, their total contributions sum to at least the desired lower bound, $\sum_{h=1}^K \sum_{i=1}^{n_h} n_{hi} w_{hi}$, minus the *timeout loss*: $\sum_{h=2}^K \sum_{i=1}^{n_h} \tau_{hi} w_{hi}$.

To bound the timeout loss by $1/L_K$ times the desired lower bound, we observe, for $h \geq 2$, that

$$\sum_{i=1}^{n_h} \tau_{hi} w_{hi} \leq \frac{1}{L_K} \sum_{j=1}^{n_{h-1}} n_{h-1,j} w_{h-1,j}, \quad (2)$$

because, within each h -phase i , each of the τ_{hi} $(h-1)$ -phases that times out contributes one of the w_{hi} 's to the left-hand sum, while its corresponding contribution to the right-hand sum, $n_{h-1,j} w_{h-1,j} = L_{h-1} w_{h-1,j}$ is, by 1.4 (ii), at least $L_K w_{hi}$.

Summing (2) over $h \geq 2$, the timeout loss is at most $1/L_K$ times the desired lower bound. \square

Lemmas 1.3, 1.5 and 1.6 together with the observation that $w_{k1} n_{k1} = L_K$, imply (by algebra) that the cost of A divided by the cost of OPT is at least $(1 - O(K/L_K))K$. \square

2 Schedules for BMC_f as binary search trees

This section proves Theorem 2: for any instance \mathcal{I} of BMC_f , the schedules are isomorphic to n -node binary search trees. Fix any instance \mathcal{I} of BMC_f . Let n be the length of \mathcal{I} .

Definition 2.1. *A tree for \mathcal{I} is any n -node binary search tree T holding keys $\{1, 2, \dots, n\}$.*

*Define $\text{latency}(T) = \max_{t=1}^n 1 + \text{right_depth}_T(t)$.*²

Define $\text{cost}_f(T) = \sum_{t=1}^n \ell_t (1 + \text{left_depth}_T(t)) + r_t f(1 + \text{right_depth}_T(t))$.

Recall that, given a schedule σ , k_t denotes the stack size that σ yields at time t .

Theorem 2. *There is a bijection ϕ between the schedules for \mathcal{I} and the trees for \mathcal{I} . Further, for any schedule σ and its tree $T = \phi(\sigma)$, for each $t \in [1, n]$, $k_t = 1 + \text{right_depth}_T(t)$, and the number of times σ merges the file inserted at time t (directly or indirectly) is $1 + \text{left_depth}_T(t)$. Hence, the bijection preserves latency and cost.*

Before proving Theorem 2, to develop intuition, we state a natural recurrence relation for $\text{OPT}(\mathcal{I})$. The reader can focus on LINEAR BMC ($f(k) = k$).

²The path from the root to the node with key t has $\text{left_depth}_T(t)$ left children and $\text{right_depth}_T(t)$ right children.

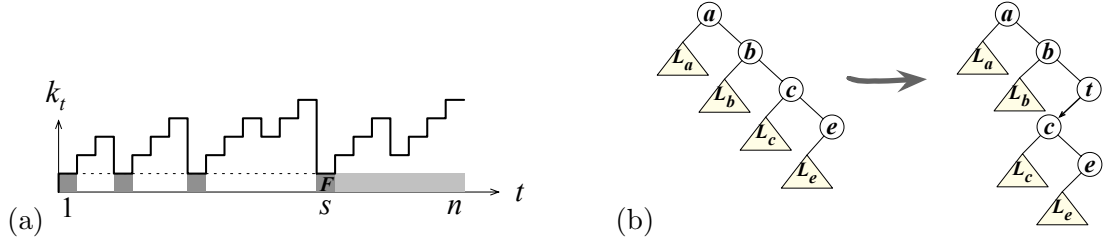


Figure 2: (a) File F is untouched after the last time s s.t. $k_s = 1$. (b) Maintaining T in the online setting.

Definition 2.2. Define $f_d(k) = f(k + d) - f(d)$ if $d \geq 1$ and $f_0 = f$. For each (i, j, d) , let $\mathcal{I}_d[i, j]$ be the BMC_{f_d} instance with read-cost function f_d and input sequence $(\ell_i, r_i), (\ell_{i+1}, r_{i+1}), \dots, (\ell_j, r_j)$.

Let $\text{OPT}_d[i, j]$ denote the minimum cost of any schedule to $\mathcal{I}_d[i, j]$. For $i > j$, let $\text{OPT}_d[i, j] = 0$.

Let $\ell[i, j] = \sum_{h=i}^j \ell_h$, and $r[i, j] = \sum_{h=i}^j r_h$.

Lemma 2.1 (recurrence relation for BMC_f). $\text{OPT}(\mathcal{I}) = \text{OPT}_0[1, n]$ and, for $1 \leq i \leq j \leq n$ and $d \geq 0$,

$$\text{OPT}_d[i, j] = \min_{s=i \dots j} \text{OPT}_d[i, s-1] + \ell[i, s] + r[s, j]f_d(1) + \text{OPT}_{d+1}[s+1, j]. \quad (3)$$

Proof. Consider any schedule σ for $\mathcal{I}_0[1, n]$. As shown in Figure 2(a), let $s \in [1, n]$ be the last time that σ has stack size 1 ($k_s = 1$). The schedule σ decomposes into three parts as follows: (i) during interval $[1, s-1]$, a schedule for $\mathcal{I}_0[1, s-1]$; (ii) at time s , a merge of all files into a single file, say, F , at merge cost $\ell[1, s]$; (iii) during interval $[s+1, n]$, a schedule for $\mathcal{I}_1[s+1, n]$, during which F remains untouched at the bottom of the stack, so that F contributes read cost $r[s, n]f(1)$.

Conversely, any $s \in [1, n]$, schedule for $\mathcal{I}_0[1, s-1]$ and schedule for $\mathcal{I}_1[s+1, n]$ yield a schedule for $\mathcal{I}_0[1, n]$. This gives Recurrence (3) for $\text{OPT}_0[1, n]$. The general case is similar. \square

Proof of Theorem 2. Fix any schedule σ for \mathcal{I} . Construct the corresponding tree $T = \phi(\sigma)$ by following the inductive structure implicit in the proof of 2.1 — take s to be the last time that σ makes $k_s = 1$ (see Figure 2(a)), make s the key of the root, then recurse on intervals $[1, s-1]$ and $[s+1, n]$, respectively, to build T 's left and right subtrees. An easy inductive argument shows that every node has the desired left and right depth. Given any tree T for \mathcal{I} , the construction can be inverted to construct a corresponding schedule σ , completing the proof. \square

Corollary 2. There is an $O(n^4)$ -time dynamic-programming algorithm for offline BMC_f . For $\text{BMC}_{\leq K}$ and LINEAR BMC, the time reduces to $O(Kn^3)$ and $O(n^3)$, respectively.

Online BMC_f is equivalent to building a binary search tree online. Via Theorem 2, online BMC_f has a natural interpretation as the following online problem. Given a BMC_f instance \mathcal{I} , as each pair (ℓ_t, r_t) is revealed, the algorithm A must maintain a tree T for $\mathcal{I}[1, t]$. At time $t = 1$, the tree T is a single node with key 1. At each time $t > 1$, A must insert a new node with key t into T , without changing the relations of nodes already in T . That is, A either appends the new node to the right spine (as the right child of the bottom node), or inserts the new node into the right spine above some node c , moving c to the left child of the new node (the new node has no right child), as shown in Figure 2(b). The goal is to minimize $\text{cost}_f(T)$.

By a straightforward induction, valid sequences of insertions correspond to valid sequences of compactions. The current tree T at time t corresponds (via Theorem 2) to the schedule of compactions over $[1, t]$. The nodes along the right spine of T correspond to the files in the stack at time t . We summarize this as follows:

Lemma 2.2. The c -competitive online algorithms for the problem above correspond to the c -competitive online algorithms for BMC_f .

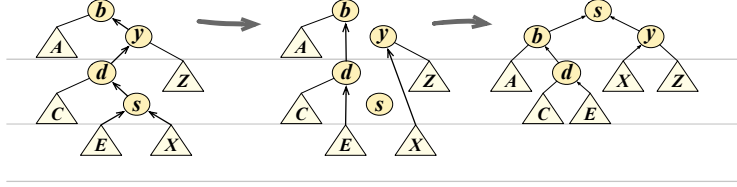


Figure 3: In the proof of Lemma 3, moving s to the root to transform T^* into T' .

3 Worst-case analysis of linear BMC

Definition 3. In any tree T for \mathcal{I} , let T_t , L_t , and R_t denote, respectively, the subtree with root key t and its left and right subtrees. In any subtree T_t , the keys in T_t form an interval $[i, j]$. Let $\ell[T_t] = \ell[i, j] = \sum_{t=i}^j \ell_t$ and $r[T_t] = r[i, j] = \sum_{t=i}^j r_t$. (Define $\ell[L_t] = r[R_t] = 0$ for empty L_t , R_t .)

Lemma 3 (lower bound on OPT for LINEAR BMC). For any instance \mathcal{I} of LINEAR BMC, any schedule σ , and its tree $T = \phi(\sigma)$,

- (i) $\text{cost}(T) = \sum_{t=1}^n \ell_t + r_t + \ell[L_t] + r[R_t]$, and
- (ii) $\text{OPT}(\mathcal{I}) \geq \text{cost}(T) - \sum_{t=1}^n \max\{\ell[L_t], r[R_t]\} = \sum_{t=1}^n \ell_t + r_t + \min\{\ell[L_t], r[R_t]\}$.

Proof. Part (i) follows by calculation from the definition of $\text{cost}(T)$. To prove Part (ii), let T^* be a tree of cost $\text{OPT}(\mathcal{I})$. Transform T^* into T , without increasing the cost by much, as follows. Let s be the root of T . First transform T^* into a tree T' with s at the root. In T^* , for each node $x < s$, change the parent to the first ancestor less than s (if any). For each node $x > s$, change the parent to the first ancestor greater than s (if any). This splits $T^* \setminus \{s\}$ into a tree $T'_<$ for $[1, s-1]$ and a tree $T'_>$ for $[s+1, n]$, as shown in Figure 3. Make s the root of T' , with $T'_<$ as the left subtree and $T'_>$ as the right subtree. This defines T' . To complete the transformation, transform the left and right subtrees of T' recursively into, respectively, the left and right subtrees of T .

How are left and right depths of nodes changed in the transformation from T^* to T' ? If the root of T^* is smaller than s (as in Figure 3) then the only depths that may increase are the left depths of nodes in the left subtree of T' , which increase by at most 1. Hence, $\text{cost}(T') \leq \text{cost}(T^*) + \ell[L_t]$. Similarly, if the root of T^* is larger than s , then $\text{cost}(T') \leq \text{cost}(T^*) + r[R_t]$. It follows that $\text{cost}(T') \leq \text{cost}(T^*) + \max\{\ell[L_s], r[R_s]\}$.

By induction, transforming T' into T by recursing into T' 's two subtrees increases the cost by at most $\sum_{t \neq s} \max\{\ell[L_t], r[R_t]\}$, so the total cost increase in transforming T^* into T is at most $\sum_{t=1}^n \max\{\ell[L_t], r[R_t]\}$. It follows that $\text{OPT}(\mathcal{I}) = \text{cost}(T^*) \geq \text{cost}(T) - \sum_{t=1}^n \max\{\ell[L_t], r[R_t]\}$. \square

For intuition, note that Lemma 3 gives a fast offline 2-approximation algorithm:

Corollary 3. There is an $O(n)$ -time, offline 2-approximation algorithm for LINEAR BMC.

Proof. Fix an instance \mathcal{I} , schedule σ and its tree T . Say node t in T is *balanced* if $|\ell[L_t] - r[R_t]| \leq \ell_t + r_t$.

By Lemma 3(i), $\text{cost}(T) = \sum_{t=1}^n \ell_t + r_t + \ell[L_t] + r[R_t]$. Comparing this sum term-by-term with the lower bound on $\text{OPT}(T)$ from Lemma 3(ii), it follows that if every node in T is balanced, then $\text{cost}(T) \leq 2 \text{OPT}(\mathcal{I})$:

$$\text{cost}(T) = \sum_{t=1}^n \ell_t + r_t + 2 \min\{\ell[L_t], r[R_t]\} + |\ell[L_t] - r[R_t]| \leq \sum_{t=1}^n 2(\ell_t + r_t + \min\{\ell[L_t], r[R_t]\})$$

To construct such a T , use binary search to find the maximum $s \in [1, n+1]$ such that $\ell[1, s-1] \leq r[s, n]$ (so $\ell[1, s] > r[s+1, n]$, this ensures s is balanced). Make s the root of T , then recurse on $[1, s-1]$ and $[s+1, n]$. \square

We note without proof that Lemma 3 and Corollary 3 extend to BMC_f for any concave f .³

Next we develop the online algorithm A . We describe A as an online algorithm for maintaining a tree T , per 2.2. To guarantee λ -competitiveness, we ensure that $\text{cost}_f(T)$ is at most λ times the lower bound T gives via Lemma 3. A maintains the following invariant on T :

$$\forall s \in [1, t]. \ell[L_s] \geq r[R_s]. \quad (4)$$

At each time t , A inserts the new node with key t as high as possible on the right spine, subject to Invariant (4). (Inserting t at the bottom of the spine is one way to maintain the invariant.)

Theorem 3 (LINEAR BMC worst-case analysis). *The online algorithm A above is $O(1)$ -competitive on those instances \mathcal{I} of LINEAR BMC such that $\ell_t = O(r_t)$ for all t .*

Proof. Fix any instance \mathcal{I} such that $\ell_t \leq \alpha r_t$ for all t (where $1 \leq \alpha = O(1)$). We use an amortized analysis to show that $\text{cost}(T)$ is always at most $1 + \alpha$ times the lower bound that T gives on OPT via Lemma 3(ii). Let $\mathcal{S}(T)$ denote the nodes in T that are on the right spine.

As A maintains T , define the *potential* of T to be

$$\Phi(T) = \sum_{x \in T} (\ell_x + r_x + r[R_x]) \times \begin{cases} 1 & x \in \mathcal{S}(T) \\ 2 & x \notin \mathcal{S}(T). \end{cases} \quad (5)$$

By inspection of Φ , Invariant (4) implies that $\Phi(T)$ is $O(1)$ times the lower bound from Lemma 3. By calculation, at time step t , the increase in $\text{cost}(T)$ is $k_t r_t + \ell_t + \ell[T_c]$, where k_t is the number of nodes on the right spine after time t and c is the node that becomes the left child of t after the insertion. (as in Figure 2(b)). To finish, we verify by calculation (using $\ell[R_c] \leq \alpha r[T_c]$) that this increase is less than $1 + \alpha$ times the increase in $\Phi(T)$. That is, $\Delta \text{cost}(T) \leq (1 + \alpha) \Delta \Phi(T)$.

Consider the insertion of node t . Recall $\text{cost}(T) = \sum_{x \in T} \ell_x + r_x + \ell[L_x] + r[R_x]$. First consider the case when t is inserted at the bottom of the right spine. Then $\text{cost}(T)$ increases by $\ell_t + k_t r_t$. The potential increases by $\ell_t + (k_t + 1) r_t$, so we are done. Otherwise, t is inserted along the right spine, with node c on the spine becoming the left child of t . Let k_t be the length of the spine after the insertion. Now,

$\Delta \Phi(T) \geq k_t r_t + \ell_t + \ell_c + r[R_c]$	Inspecting Φ , using that c leaves spine $\mathcal{S}(T)$. (6)
$\Delta \text{cost}(T) = k_t r_t + \ell_t + \ell[T_c]$	Using $L_t = T_c$ and $R_t = \emptyset$ and def'n of cost . (7)
$\ell[T_c] = \ell_c + \ell[L_c] + \ell[R_c]$	By definition of $\ell[X]$. (8)
$\ell[L_c] < r_t + r[R_c]$	By the algorithm's choice of c . (9)
$\ell[R_c] \leq \alpha r[R_c]$	By the assumption $\forall x. \ell_x \leq \alpha r_x$. (10)
$\Delta \text{cost}(T) < k_t r_t + \ell_t + \ell_c + r_t + (1 + \alpha) r[R_c]$	Transitively from (7)–(10). (11)
$\Delta \text{cost}(T) < (1 + \alpha) \Delta \Phi(T)$	Comparing (6) and (11).

\square

³Define $f_d(k) = f(k) - f(d)$ if $d > 0$, and $f_0 = f$. Let $d(t) = \text{right_depth}_T(t)$. Then (i) $\text{cost}_f(T) = \sum_{t=1}^n \ell_t + \ell[L_t] + (r_t + r[R_t]) f_{d(t)}(1)$ and (ii) $\text{OPT}(\mathcal{I}) \geq \text{cost}_f(T) - \sum_{t=1}^n \ell_t + r_t f_{d(t)}(1) + \min\{\ell[L_t], r[R_t] f_{d(t)}(1)\}$.

4 Average-case analyses of BMC_K and linear BMC

Theorem 4. $\text{BMC}_{\leq K}$ and LINEAR BMC have online algorithms A and B , respectively, that are asymptotically 1-competitive in expectation on random inputs \mathcal{I} with bounded, i.i.d. requests. Let $\mathcal{I} = \langle (\ell_t, r_t) \rangle_t$ be a random sequence of n i.i.d. pairs from any bounded probability distribution over $\mathbb{R}_+ \times \mathbb{R}_+$. Let $(\bar{\ell}, \bar{r}) = (\mathbb{E}[\ell_t], \mathbb{E}[r_t])$ for all t . For $\text{BMC}_{\leq K}$,

$$\mathbb{E}_{\mathcal{I}}[A(\mathcal{I})] \sim \mathbb{E}_{\mathcal{I}}[\text{OPT}(\mathcal{I})] \sim \bar{\ell} K n^{1+1/K} / c_K$$

where $c_K = (K+1)/(K!)^{1/K}$ (so $c_K \rightarrow e$ for large K).

$$\mathbb{E}_{\mathcal{I}}[B(\mathcal{I})] \sim \mathbb{E}_{\mathcal{I}}[\text{OPT}(\mathcal{I})] \sim \beta n \log_2 n,$$

where β satisfies $1/2^{\bar{\ell}/\beta} + 1/2^{\bar{r}/\beta} = 1$, so $\beta = \Theta(\bar{\ell} + \bar{r}) / \ln(1 + \max(\bar{\ell}/\bar{r}, \bar{r}/\bar{\ell}))$.

We conjecture that BRB is also asymptotically 1-competitive on bounded i.i.d. inputs.

Before we prove the theorem, we prove two utility lemmas. The first characterizes optimal costs on *uniform* instances $\bar{\mathcal{I}}$, that is, $\bar{\mathcal{I}} = (\bar{\ell}, \bar{r})^n$ for some $(\bar{\ell}, \bar{r}) \in \mathbb{R}_+ \times \mathbb{R}_+$:

Lemma 4.1 (uniform instances). *Fix any $(\bar{\ell}, \bar{r}) \in \mathbb{R}_+ \times \mathbb{R}_+$. Let $\bar{\mathcal{I}} = (\bar{\ell}, \bar{r})^n$.*

- (i) *For $\text{BMC}_{\leq K}$, $\text{OPT}(\bar{\mathcal{I}}) \sim \bar{\ell} K n^{1+1/K} / c_K$, for c_K as defined in Theorem 4.*
- (ii) *For LINEAR BMC, $\text{OPT}(\bar{\mathcal{I}}) \sim \beta n \log n$, for β such that $1/2^{\bar{\ell}/\beta} + 1/2^{\bar{r}/\beta} = 1$.*

The value of β is $\Theta(\max\{\bar{\ell}/\log \bar{\ell}/\bar{r}, \bar{r}/\log \bar{r}/\bar{\ell}\})$.

Proof. By Theorem 2, the optimal costs equal the costs of optimal n -node binary search trees under an appropriate cost function. For uniform instances, these cost functions are well-studied, and optimal costs are known to asymptotically equal these quantities (e.g. [3, 10, 12]). Here are the details.

(i) For the read-cost function f for $\text{BMC}_{\leq K}$, the tree T for \mathcal{I} that minimizes $\text{cost}_f(T)$ has right-depth at most $K-1$, and, subject to that constraint, has n nodes chosen to minimize total left-depth. This T is well understood (e.g. [3]). T has maximum left-depth d , where, by calculation, d is minimum subject to $\binom{K+d}{K} \geq n$, so $d \sim (K!n)^{1/K}$. T has total left-depth $\sim \frac{K}{K+1} dn$. By Theorem 2, $\text{OPT}(\mathcal{I}) \sim \frac{K}{K+1} dn = K n^{1+1/K} / c_K$.

(ii) For the read-cost function f for LINEAR BMC, the tree for $\bar{\mathcal{I}}$ that minimizes $\text{cost}_f(T)$ corresponds to an optimal *lopsided alphabetic code* — a sequence of n distinct (and ordered) binary codewords C_1, C_2, \dots, C_n , where the cost of C_t is $\bar{\ell}$ times the number of zeros in C_t plus \bar{r} times the number of ones. Such codes are well-studied (e.g., [10, 12]), and have minimum total cost $\sim \beta n \log n$. By Theorem 2, $\text{OPT}(\mathcal{I}) \sim \beta n \log n$. \square

As an aside, this approach extends to other special cases. For example, consider any “proportional” instance \mathcal{I} of LINEAR BMC such that, for some $\alpha > 0$, each pair (ℓ_t, r_t) satisfies $\ell_t = \alpha r_t$. Then $\text{OPT}(\mathcal{I}) \sim \beta r[1, n] H(p)$, where $H(p)$ is the entropy of the distribution p such that $p_t = r_t/r[1, n]$, and β is such that $1/2^{\alpha/\beta} + 1/2^{1/\beta} = 1$ [10].

Next we prove that one can replace uniform requests by bounded, i.i.d. requests without changing optimal asymptotic costs. For the remainder of the proof, let \mathcal{I} , $\bar{\ell}$, and \bar{r} be as in Theorem 4. Let $\bar{\mathcal{I}} = \mathbb{E}[\mathcal{I}] = (\bar{\ell}, \bar{r})^n$. Take $\delta = 100 U \log(n) / n \epsilon^2$, where $U \geq \max_t \max(\ell_t/\bar{\ell}, r_t/\bar{r})$ gives an absolute upper bound on lengths and read costs from the distribution, and $\epsilon \rightarrow 0$ slowly as $n \rightarrow \infty$ (e.g. $\epsilon = 1/\log n$), so $\epsilon = o(1)$. Call intervals $[i, j]$ of length at least δn *large*, and the rest *small*. Say that \mathcal{I} *behaves* if $\ell[i, s] + r[s, j] \geq (1-\epsilon)[(s-i+1)\bar{\ell} + (j-s+1)\bar{r}]$ and $\ell[i, j] \geq (1-\epsilon)(j-i+1)\bar{\ell}$ for every large interval $[i, j] \subseteq [1, n]$ and every $s \in [i, j]$.

Lemma 4.2. \mathcal{I} behaves with probability $1 - o(n^{-10})$.

Proof. This follows from a standard Chernoff bound and the naive union bound. Here are the details. Consider any large $[i, j]$ and $s \in [i, j]$. By a standard Chernoff bound, using $j - i + 1 \geq \delta n$,

$$\Pr[\ell[i, j] \leq (1 - \epsilon)(j - i + 1)] \leq \exp(-\epsilon^2(j - i + 1)\bar{\ell}/(3U/\bar{\ell})) \leq \exp(-33 \log n) = n^{-33}.$$

Likewise, $\Pr[\ell[i, s] + r[s, j] \leq (1 - \epsilon)[(s - i + 1)\bar{\ell} + (j - s + 1)\bar{r}]$ is at most n^{-33} . Since there are at most n^3 triples (i, s, j) , the probability that \mathcal{I} misbehaves is at most $2n^{-30} = o(n^{-10})$. \square

Lemma 4.3. For both $\text{BMC}_{\leq K}$ and LINEAR BMC, $E_{\mathcal{I}}[\text{OPT}(\mathcal{I})] \sim \text{OPT}(\bar{\mathcal{I}})$.

Proof. Let $\bar{\sigma}$ be an optimal schedule for $\bar{\mathcal{I}}$. Then $E[\text{OPT}(\mathcal{I})] \leq E[\bar{\sigma}(\mathcal{I})] = \bar{\sigma}(E[\mathcal{I}]) = \bar{\sigma}(\bar{\mathcal{I}}) = \text{OPT}(\bar{\mathcal{I}})$. (The first equality holds by linearity of expectation, as $\bar{\sigma}(\mathcal{I})$ is a linear function of $\mathcal{I} = \langle (\ell_t, r_t) \rangle_t$.) This shows $E[\text{OPT}(\mathcal{I})] \leq \text{OPT}(\bar{\mathcal{I}})$. It remains to show $E_{\mathcal{I}}[\text{OPT}(\mathcal{I})] \geq (1 - o(1)) \text{OPT}(\bar{\mathcal{I}})$.

First we prove the claim for LINEAR BMC. For LINEAR BMC Recurrence (3) simplifies to

$$\text{OPT}[i, j] = \min_{s=i \dots j} \text{OPT}[i, s - 1] + \ell[i, s] + r[s, j] + \text{OPT}[s + 1, j]. \quad (12)$$

Assume that \mathcal{I} behaves. Then (by induction on the recurrences) $\text{OPT}[i, j] \geq (1 - \epsilon) \text{LB}[i, j]$, where

$$\text{LB}[i, j] = \min_{s=i \dots j} \text{LB}[i, s - 1] + (s - i + 1)\bar{\ell} + (j - s + 1)\bar{r} + \text{LB}[s + 1, j] \quad (13)$$

for large intervals $[i, j]$ and $\text{LB}[i, j] = 0$ for small $[i, j]$. To finish we show $\text{LB}[1, n] \geq (1 - o(1)) \text{OPT}(\bar{\sigma})$. Let T be the recursion tree for Recurrence (13) for $\text{LB}[1, n]$, interpreted as a binary search tree on keys $[1, n]$ as in the proof of Thm. 2. In T , for each maximal subtree S whose interval $[i, j]$ is small, replace S by the optimal subtree for $\bar{\mathcal{I}}[i, j]$. Let T' be the resulting tree. Using T' as a solution (schedule) for $\text{OPT}(\bar{\mathcal{I}})$, and letting S range over the subtrees introduced into T' , $\text{OPT}(\bar{\mathcal{I}}) \leq \text{cost}(T') = \text{LB}[1, n] + \sum_S \text{cost}(S)$.

The number of subtrees S is at most $n/\delta n = 1/\delta$. Each has $\text{cost}(S) = O(\beta \delta n \log(\delta n))$ (Theorem 4(ii)), so $\sum_S \text{cost}(S)$ is $O((1/\delta)(\beta \delta n \log(\delta n)))$, which is $o(\text{OPT}(\bar{\mathcal{I}}))$, as $\delta n = \log^{O(1)} n$.

Hence $E_{\mathcal{I}}[\text{OPT}(\mathcal{I})] \geq \Pr[\mathcal{I} \text{ behaves}](1 - o(1)) \text{OPT}(\bar{\mathcal{I}}) \sim \text{OPT}(\bar{\mathcal{I}})$.

To finish, we prove the claim for $\text{BMC}_{\leq K}$. We show $E_{\mathcal{I}}[\text{OPT}(\mathcal{I})] \geq (1 - o(1)) \text{OPT}(\bar{\mathcal{I}})$ for $\text{BMC}_{\leq K}$. The idea is the same as for LINEAR BMC. Define $\text{LB}_0[1, n]$ by recurrence

$$\text{LB}_d[i, j] = \min_{s=i \dots j} \text{LB}_d[i, s - 1] + \text{LB}_{d+1}[s + 1, j] + \begin{cases} \ell[i, s] & \text{if } [i, s] \text{ large,} \\ 0 & \text{otherwise,} \end{cases}$$

for $d < K$ and $[i, j]$ large, while $\text{LB}_K[i, j] = \infty$ for $i \leq j$, and otherwise $\text{LB}_d[i, j] = 0$ for $[i, j]$ small. As in the proof sketch, if \mathcal{I} behaves, then $\text{OPT}(\mathcal{I}) \geq (1 - \epsilon) \text{LB}_0[1, n]$. Let T be the recurrence tree for $\text{LB}_0[1, n]$. Interpret T as a solution for $\bar{\mathcal{I}}$, and, for each maximal subtree S for a subproblem $\bar{\mathcal{I}}_d[i, j]$ where $[i, j]$ is small, replace S by the optimal subtree for $\bar{\mathcal{I}}_d[i, j]$. Call the resulting tree T' . Then, interpreting T' as a solution for $\bar{\mathcal{I}}$, and letting S range over the subtrees introduced into T' , $\text{OPT}(\bar{\mathcal{I}}) \leq \text{cost}(T') \leq \text{LB}_0[1, n] + 2 \sum_S \text{cost}(S)$.

(The factor of 2 accounts for each term $\ell[i, s]$ that can be “missing” for the parent of each subtree S , in the recurrence for $\text{LB}_d[i, j]$.) There are at most $n/\delta n = 1/\delta$ subtrees S , each with $\text{cost}(S) = O((\delta n)^2)$, so $\sum_S \text{cost}(S)$ is $O(\delta n^2) = O(n \log^{O(1)} n) = o(\text{OPT}(\bar{\mathcal{I}}))$. \square

Finally we prove Theorem 4.

Proof. First consider the case when n and the distribution p are known. On input \mathcal{I} , have A ignore the input, and do merges exactly as $\text{OPT}(\bar{\mathcal{I}})$ would. Then as a function of the input vector \mathcal{I} , the function $\mathcal{I} \mapsto A(\mathcal{I})$ is linear. By linearity of expectation, $\mathbb{E}[A(\mathcal{I})] = A(\bar{\mathcal{I}}) = \text{OPT}(\bar{\mathcal{I}})$, which asymptotically equals $\mathbb{E}[\text{OPT}(\mathcal{I})]$ by 4.3.

To handle the case when p and n are not known, use the fact that the optimal schedule for $\bar{\mathcal{I}}$ depends only on two parameters: $\bar{\ell}$ and \bar{r} . At each time t that is a power of two, start a new *phase*: merge all files into one file F , then, during the phase $[t, 2t - 1]$ ignore F completely and follow the optimal schedule for $(\bar{\ell}', \bar{r}')^t$, where $\bar{\ell}'$ and \bar{r}' are the average file length and read rate so far.

The total cost for the merges at the start of each phase and for the bottom stack slot is $O(\ell[1, n] + r[1, n]) = o(\text{OPT}(\bar{\mathcal{I}}))$. We bound the remaining cost. Take δ , ϵ , and U as earlier defined. The cumulative cost of the online algorithm through the phase containing time δn is $O(U\bar{\ell}(\delta n)^2) = o(\text{OPT}(\bar{\mathcal{I}}))$ (using $\delta n = O(\log n)$ and $\text{OPT}(\bar{\mathcal{I}}) = \Omega(n \log n)$). After that time, with high probability, the estimates of $\bar{\ell}$ and \bar{r} are all $(1 \pm \epsilon)$ -accurate, so, phase by phase, the expected cost of the online algorithm tracks the cost of $\text{OPT}((\bar{\ell}, \bar{r})^t)$ within a $1 + o(1)$ factor. (To handle phase $[t, 2t - 1]$, the algorithm follows a static schedule, say σ , for $\text{OPT}((\bar{\ell}', \bar{r}')^t)$, and incurs expected cost $\sigma((\bar{\ell}, \bar{r})^t) \leq (1 + \epsilon)\sigma((\bar{\ell}', \bar{r}')^t) \leq (1 + \epsilon)\text{OPT}((\bar{\ell}', \bar{r}')^t) \leq (1 + \epsilon)^2 \text{OPT}((\bar{\ell}, \bar{r})^t)$.) Hence, the expected cost of the algorithm after the phase containing time δn is $(1 + o(1)) \text{OPT}(\bar{\mathcal{I}}) = (1 + o(1)) \mathbb{E}[\text{OPT}(\mathcal{I})]$. \square

5 Benchmarks

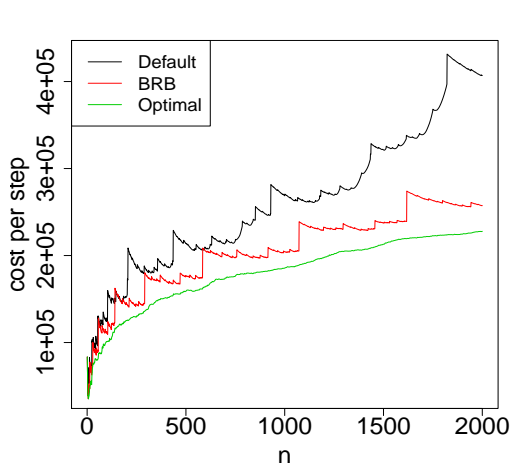
For $\text{BMC}_{\leq K}$, we test BRB and Google's DEFAULT algorithm (*merge minimally, subject to the constraint that each file remains as large as all files above it combined*). For LINEAR BMC we test the algorithms from Theorem 3 and Theorem 4. The inputs are sequences with read costs i.i.d. from an exponential distribution and file lengths i.i.d. from a log-normal distribution. We let μ and v denote the mean and variance of the underlying normal distribution. When computationally feasible, we also test OPT. Each plot plots average cost *per time step* (that is, total cost divided by n) versus n , for several algorithms on one input.

Results for $\text{BMC}_{\leq K}$. Recall that for $\text{BMC}_{\leq K}$, we expect OPT to cost about $\bar{\ell} K n^{1/K}/e$ (per time step). We hope that BRB costs about the same. On uniform instances, by calculation DEFAULT costs about $\bar{\ell} n/(2 \cdot 3^{K-1})$ per time step. We expect DEFAULT to have roughly this cost on i.i.d. instances as well. As a consequence, we expect that BRB should substantially outperform DEFAULT for large n , say, for $n \geq K 3^K$. We do see this. We also see that, in general, BRB is close to OPT, and better than DEFAULT even for small n . See Fig. 4 for an example.

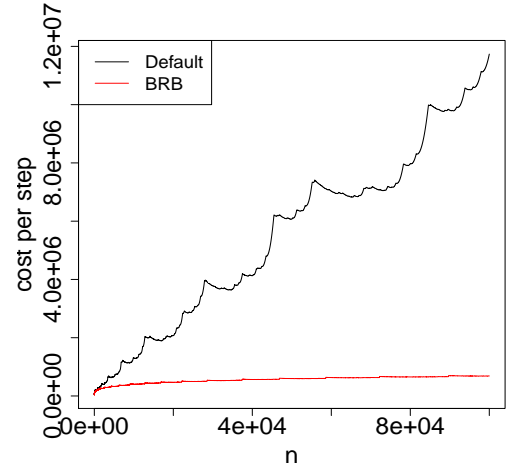
Results for LINEAR BMC. Recall that for LINEAR BMC, we expect OPT to cost about $\beta \log n$ (per time step), where $1/2^{\bar{\ell}/\beta} + 1/2^{\bar{r}/\beta} = 1$. We hope that our online algorithms achieve cost near this. (We know that the LINEAR BMC algorithm from Theorem 4 does *asymptotically*.) We find that they do, even for small n , except that when $\bar{\ell}/\bar{r}$ is large, the algorithm from Theorem 3 doesn't do as well. See Fig. 5 for an example.

6 Acknowledgements

Thanks to Mordecai Golin and Vagelis Hristidis for useful discussions.

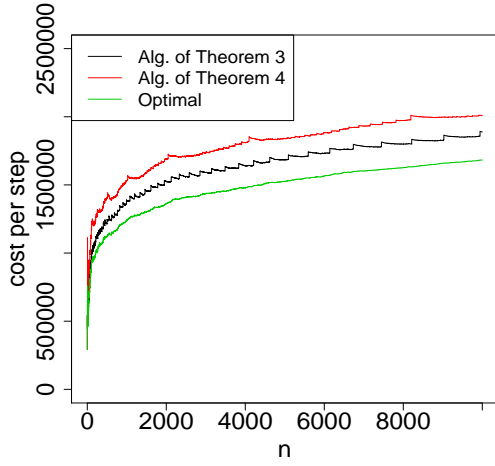


(a) $\text{BMC}_{\leq K}$ with $K = 5, n \leq 2000$

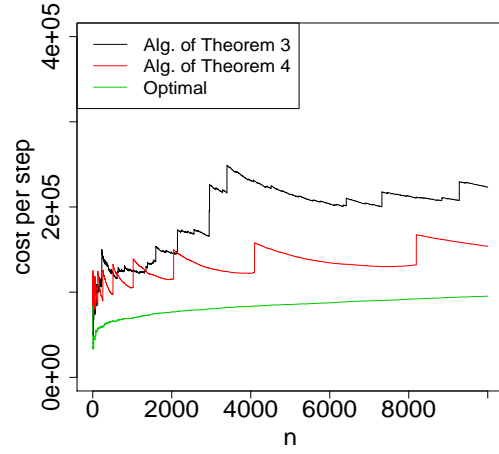


(b) $\text{BMC}_{\leq K}$ with $K = 5, n \leq 100,000$

Figure 4: An instance with $\mu = 10, v = 1$, so typically $\ell_t \in [e^9, e^{11}]$.



(a) LINEAR BMC with $\bar{\ell}/\bar{r} = .1, n \leq 10,000$



(b) LINEAR BMC with $\bar{\ell}/\bar{r} = 100, n \leq 10,000$

Figure 5: Instances with $\mu = 10, v = 1$, and (a) $\bar{\ell}/\bar{r}$ small, and (b) $\bar{\ell}/\bar{r}$ large.

References

- [1] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. Borkar, Y. Bu, M. Carey, I. Cetindil, M. Cheelangi, K. Faraaz, et al. AsterixDB: A scalable, open source BDMS. *Proceedings of the VLDB Endowment*, 7(14):1905–1916, 2014.
- [2] L. Arge and N. Zeh. External-memory algorithms and data structures. In M. J. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook*, pages 10–10. Chapman & Hall/CRC, 2010.
- [3] J. L. Bentley and D. J. Brown. A general class of resource tradeoffs. *Journal of Computer and System Sciences*, 25(2):214–238, Oct. 1982.
- [4] R. Cattell. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.
- [5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.
- [6] D. Choy and C. Wong. Construction of optimal α — β leaf trees with applications to prefix code and information retrieval. *SIAM Journal on Computing*, 12(3):426–446, Aug. 1983.
- [7] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. Spanner: Google’s globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.
- [8] L. George. *HBase: the definitive guide*. O’Reilly Media, 2011.
- [9] M. Ghosh, I. Gupta, S. Gupta, and N. Kumar. Fast compaction algorithms for NoSQL databases. Technical report, University of Illinois, Dept. of Computer Science, Apr. 2015.
- [10] M. Golin and J. Li. More efficient algorithms and analyses for unequal letter cost prefix-free coding. *IEEE Transactions on Information Theory*, 54(8):3412–3424, Aug. 2008.
- [11] D. Judd. Scale out with HyperTable. *Linux magazine*, August 7th, 2008.
- [12] S. Kapoor and E. M. Reingold. Optimum lopsided binary trees. *J. ACM*, 36(3):573–590, July 1989.
- [13] J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, P. Michaleas, J. Mullen, A. Prout, A. Reuther, A. Rosa, and C. Yee. Achieving 100,000,000 database inserts per second using Accumulo and D4m. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, Sept. 2014.
- [14] A. Khetrpal and V. Ganesh. HBase and Hypertable for large scale distributed storage systems. *Dept. of Computer Science, Purdue University*, pages 22–28, 2006.
- [15] S. Patil, M. Polte, K. Ren, W. Tantisiroj, L. Xiao, J. Lpez, G. Gibson, A. Fuchs, and B. Rinaldi. YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 9. ACM, 2011.
- [16] E. Redmond and J. R. Wilson. *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*. Pragmatic Bookshelf, 2012.
- [17] M. Sniedovich. OR/MS Games: 4. The joy of egg-dropping in Braunschweig and Hong Kong. *INFORMS Transactions on Education*, 4(1):48–64, 2003.
- [18] C. Strauch. NoSQL databases. *Lecture Notes, Stuttgart Media University*, 2011.
- [19] J. S. Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Comput. Surv.*, 33(2):209–271, June 2001.